گزارش تمرین سوم

آشنایی با سیستم عامل اندروید و استفاده از سنسورهای تلفن همراه

سیستم‌های نهفته بی‌درنگ

اعضای گروه:

مبینا شاه‌بنده، امید بداقی، دیار محمدی، غزل کلهر

بهار ۱۴۰۰

فهرست مطالب

توضیح کد

هر دو سنسور در یک فایل apk پیاده سازی شده اند (امتیازی) و هر دو نیز به علت تشابه کد در یک activity نوشته شده اند و با استفاده از intent نوع سنسور به game activity فرستاده شده است.

MainActivity

این activity، activity اصلی است که هنگام باز کردن بازی اجرا می شود. برا این activity دو فایل وجود دارد که یک view و یکی مدل آن است. فایل view به صورتی نوشته شده که در آن عنوان بازی و دو دکمه gyroscope و gravity نمایش داده شوند. در فایل مدل که به زبان جاوا نوشته شده نوع سنسور بر اساس فشردن هر یک از دو دکمه تعیین شده و سپس کاربر به game activity هدایت می شود.

GameActivity

این activity، activity بازی است که در آن توپ حرکت می کند. برا این activity دو فایل وجود دارد که یک view و یکی مدل آن است. فایل view به صورتی نوشته شده که در آن توپ و دکمه پرش توپ نمایش داده شوند. در فایل مدل که به زبان جاوا نوشته شده بر اساس نوع سنسور، یک Listener روی سنسور ساخته می شود که در آن تغییر مقدار سنسور پردازش می شود. در ادامه توضیح مفصل توابع این فایل آمده است.

```
protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.activity_game);

     getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN
, WindowManager.LayoutParams.FLAG_FULLSCREEN);

      ballImageView = findViewById(R.id.image_ball);
```

```
        Pair displaySize = getDisplaySize();
        float ballRadius = dpTopx(GameConfig.BALL_RADIUS);


        _3dVector randomPosition =
RandomGenerator.random3dVector(dpTopx(GameConfig.JUMP_BTN_SIZE),
            (int)displaySize.first-(2*ballRadius),
            0, (int)displaySize.second-(2*ballRadius),
            0, 0);
        ball = new Ball(randomPosition,
            new _3dVector(0, 0, 0),
            new _3dVector(0, 0, 0),
            ballImageView,
            displaySize,
            ballRadius,
            dpTopx(GameConfig.JUMP_BTN_SIZE));
        final View playButton = findViewById(R.id.button_play);
        playButton.setOnClickListener(new View.OnClickListener() {
          @Override
```

```
    public void onClick(View view) {
      gameStarted = true;
      playButton.setVisibility(View.GONE);
    }
  });
  final View jumpButton = findViewById(R.id.button_jump);
  jumpButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      ball.generateRandomVelocity();
    }
  });
  theta = new _3dVector(0, 0, 0);
  Timer timer = new Timer();
  timer.schedule(new TimerTask() {
    @Override
    public void run() {
```

```
            ball.handleSensorEvent(theta,
                    sensor,
                    ((double) GameConfig.REFRESH_RATE) / 1000);
            }
        }, 0, GameConfig.REFRESH_RATE);
        initializeSensors();
    }
```

در این تابع که هنگام ساختن این activity اجرا می شود، ابتدا view ای که ساخته بودیم را به عنوان محتوای این activity قرار می دهیم. سپس حالت fullscreen را فعال کرده و بعد توپ را می سازیم. مکان توپ باید یک مکان تصادفی باشد که از صفحه خارج نشود. با فشردن دکمه شروع بازی باید این دکمه مخفی شود و Listener سنسورها شروع به کار کند. با فشردن دکمه پرش، سرعتی تصادفی به سمت بالا به توپ داده می شود. در ادامه ی کد، تایمری تنظیم شده تا مکان توپ در هر 16 میلی ثانیه بر اساس سنسورها به روز رسانی شود. شرح علت استفاده از این تایمر در پاسخ به سوال 3 آمده است.

```
private void initializeSensors() {
    sensorManager =
(SensorManager)getSystemService(SENSOR_SERVICE);
    gravitySensor =
sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
    gyroscopeSensor =
sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

```
    sensor = (GameConfig.sensor)
getIntent().getExtras().get("sensor");
    if (sensor == GameConfig.sensor.GYROSCOPE)
      sensorManager.registerListener(listener, gyroscopeSensor,
SensorManager.SENSOR_DELAY_FASTEST);
    else
      sensorManager.registerListener(listener, gravitySensor,
SensorManager.SENSOR_DELAY_FASTEST);
  }
  private SensorEventListener listener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
      if (gameStarted) {
        double deltaT = (sensorEvent.timestamp - lastEventTimestamp)
/ 1e9;
        if (deltaT > 0) {
          theta = new _3dVector(sensorEvent.values[0],
              sensorEvent.values[1],
```

```
                sensorEvent.values[2]);

        }

        lastEventTimestamp = sensorEvent.timestamp;

      }

    }

    @Override

    public void onAccuracyChanged(Sensor sensor, int i) {

    }

  };
```

همانطور که در بالا توضیح داده شد، بر اساس سنسوری که در Main Activity توسط کاربر انتخاب شد، Listener سنسورها ساخته می شود. حال کاری که این Listener می کند این است که مقادیر دریافت شده از سنسورها را به روز رسانی می کند تا در تابعی که به صورت متناوب در توپ فراخوانی می شود استفاده شوند. دو تابع دیگر توابع کمکی برای گرفتن سایز صفحه و ... هستند.

Ball

این فایل قسمت اصلی بازی است که در آن تمامی حرکات توپ ساخته می شود. در ادامه توابع آن مفصلا شرح داده شده است.

```
private _3dVector getNextPosition(double deltaT) {

        _3dVector amountToAdd1 =
acceleration.multiplyVectorByNum(0.5*(Math.pow(deltaT, 2)));
```

```
        _3dVector amountToAdd2 =
velocity.multiplyVectorByNum(deltaT);

        amountToAdd1.vectorAddition(amountToAdd2);

        return new _3dVector(position.x + amountToAdd1.x,

            position.y + amountToAdd1.y,

            position.z + amountToAdd1.z);

    }
```

این تابع موقعیت بعدی توپ را بر اساس سرعت و شتاب بدست می آورد اما ذخیره نمی کند. علت استفاده از این تابع، تشخیص برخورد با دیواره ها قبل از رخداد آن است.

```
private void updateVelocity(double deltaT) {

        _3dVector amountToAdd =
acceleration.multiplyVectorByNum(deltaT);

        velocity.vectorAddition(amountToAdd);

    }
```

در این تابع سرعت بر اساس شتاب به روز رسانی می شود.

```
private void updateAcceleration(_3dVector F) {

        acceleration.x = (F.x / GameConfig.BALL_WEIGHT) *
GameConfig.ACCELERATION_FACTOR;
```

```
        acceleration.y = (F.y / GameConfig.BALL_WEIGHT) *
GameConfig.ACCELERATION_FACTOR;

    }
```

در این تابع شتاب بر اساس نیروها به روز رسانی می شود. علت ضرب در ACCELERATION_FACTOR این است که حرکت توپ بدون استفاده از این ضریب (که 20 تعیین شده است) بسیار کند بود. بنابراین برای نمایش بهتر حرکت توپ این ضریب در شتاب آن ضرب شده است.

```
public void generateRandomVelocity() {

        double randomFloat = Math.random();

        int randomSign = (randomFloat > 0.5) ? 1 : -1;

        velocity =
RandomGenerator.random3dVector(GameConfig.RANDOM_VELOCITY_LOW,

            GameConfig.RANDOM_VELOCITY_HIGH,

            GameConfig.RANDOM_VELOCITY_LOW,

            GameConfig.RANDOM_VELOCITY_HIGH, 0, 0);

        velocity.y *= randomSign;

    }
```

در این تابع سرعت تصادفی که با فشردن دکمه پرش باید ایجاد شود، ساخته می شود. سرعت در جهت y می تواند هم منفی و هم مثبت باشد اما سرعت در جهت x باید لزوما مثبت باشد تا توپ به سمت بالا برود. بنابراین علامت سرعت در جهت y نیز می تواند تصادفی باشد. بازه ای که اندازه سرعت تصادفی در آن قرار دارد، 160 تا 320

تعیین شده است (در فایل config) و این بازه بر اساس آزمون و خطا بدست آمده است. در صورتی که مقدار سرعت تصادفی از این کمتر تعیین شود، توپ با سرعت بسیار کمی به سمت بالا می رود که خواسته مسئله نیست.

```
  public boolean checkWallCollision(_3dVector position) {

        boolean xCollided = (position.x <= 0) ||

            (position.x >= displayWidth);

        boolean yCollided = (position.y <= 0) ||

            (position.y >= displayHeight);

        return xCollided || yCollided;

    }
```

این تابع و تابع بعدی برای تشخیص برخورد توپ با دیواره ها هستند. در این تابع دو نوع برخورد تشخیص داده می شود که یکی برخورد با دیواره های افقی و دیگری برخورد با دیواره های عمودی هستند. برای برخورد با دیواره ها در جهت x باید x از دیواره پایینی کمتر باشد (که مختصات این نقطه همان نقطه 0 است) و یا از دیواره بالایی بیشتر باشد (که مختصات این نقطه همان عرض صفحه نمایش است). برای برخورد با دیواره ها در جهت y باید y از دیواره سمت چپ کمتر باشد (که همان نقطه 0 است).

```
private boolean handleWallCollision(_3dVector position) {

        boolean collided = false;

        if (checkWallCollision(new _3dVector(position.x + radius,
position.y, 0))) {

            velocity.y = Math.abs(velocity.y);

            collided = true;
```

```
        }

        if (checkWallCollision(new _3dVector(position.x, position.y +
radius, 0))) {

            velocity.x = Math.abs(velocity.x);

            collided = true;

        }

        if (checkWallCollision(new _3dVector(position.x + radius,

            position.y + radius * 2, 0))) {

            velocity.y = -Math.abs(velocity.y);

            collided = true;

        }

        if (checkWallCollision(new _3dVector(position.x + radius * 2,

            position.y + radius, 0))) {

            velocity.x = -Math.abs(velocity.x);

            collided = true;

        }

        if (collided) {
```

```
            velocity =
velocity.multiplyVectorByNum(GamePhysicsConfig.kineticEnergyReductio
nFactor);

        }

        return collided;

    }
```

در این تابع همان تابع بالایی صدا می شود. حالت های مختلف برخورد شامل برخورد از یک سمت و برخورد در گوشه های جعبه در نظر گرفته می شود. علت اضافه کردن شعاع این است که مختصات توپ مختصات ابتدای آن است.

```
public void updateImgView() {

        imgView.setX((float) position.x);

        imgView.setY((float) position.y);

    }
```

در این تابع مکان view مربوط به توپ در layout تغییر می یابد.

```
public void handleSensorEvent(_3dVector vec, GameConfig.sensor
sensor, double deltaT) {

        if (sensor == GameConfig.sensor.GYROSCOPE) {

            handleGyroscopeSensorEvent(vec, deltaT);

        } else {
```

```
            handleGravitySensorEvent(vec, deltaT);
        }
        handlePhysics();
        updateVelocity(deltaT);
        _3dVector nextPosition = getNextPosition(deltaT);
        boolean collided = handleWallCollision(nextPosition);
        if (collided) {
            handlePhysics();
            updateVelocity(deltaT);
        }
        position = getNextPosition(deltaT);
        updateImgView();
    }
```

این تابع در بازه‌های زمانی تعریف‌شده اجرا می‌شود. ورودی اول آن، مقادیر خوانده‌شده توسط سنسورها، ورودی دوم نوع سنسور و ورودی سوم بازه زمانی است. حال بر اساس نوع سنسور، ابتدا زاویه محاسبه می‌شود. سپس به ترتیب نیرو، شتاب و سرعت بروزرسانی می‌شوند. حال موقعیت احتمالی بعدی توپ حساب می‌شود و برخورد آن با دیوار بررسی می‌شود. در صورتی که برخوردی به دیوار رخ داده‌باشد، طبیعتا باید شتاب و سرعت یکبار دیگر تغییر کنند. در نهایت، موقعیت توپ را تغییر می‌دهیم و تصویر آن را به روزرسانی می‌کنیم.

```
    private void handlePhysics() {
```

```
        _3dVector F = getForces();

        double N = getN();

        F = handleFriction(F, N);

        updateAcceleration(F);

    }
```

در این تابع، ابتدا نیرو در دو جهت x و y و نیروی N محاسبه می‌شوند. شرایط اصطکاک اعمال می‌شود و در نهایت شتاب بروزرسانی می‌شود.

```
    private _3dVector getForces() {

        double fX = GamePhysicsConfig.earthGravity *
GameConfig.BALL_WEIGHT * Math.sin(theta.y);

        double fY = GamePhysicsConfig.earthGravity *
GameConfig.BALL_WEIGHT * Math.sin(theta.x);

        return new _3dVector(fX, fY, 0);

    }
```

در این حالت، سینوس زاویه محاسبه می‌شود و در گرانش زمین و وزن توپ ضرب می‌شود. در واقع، سینوس زاویه ضربدر گرانش زمین، شتاب و حاصلضرب آنها، نیرو را نتیجه می‌دهد.

```
    private double getN() {

        _3dVector sinTheta = new _3dVector(Math.sin(theta.x),
Math.sin(theta.y), 0);
```

```
        double N = GamePhysicsConfig.earthGravity *
GameConfig.BALL_WEIGHT *
        Math.cos(Math.atan(sinTheta.getSize() / (Math.cos(theta.x) +
Math.cos(theta.y))));
        return N;
    }
```

برای محاسبه‌ی نیروی عمود بر سطح، از فرمول زیر استفاده می‌شود:

$$mgcos(arctan(\frac{(\sqrt{sin^2(\theta_x) + sin^2(\theta_y)}}{cos(\theta_x) + cos(\theta_y)}))$$

```
private _3dVector handleFriction(_3dVector F, double N) {
        if (((position.x == 0) ||
            (position.x == displayWidth) && Math.abs(velocity.y) <
GameConfig.BALL_STOP_SPEED) ||
            ((position.y == 0) || (position.y == displayHeight) &&
                Math.abs(velocity.x) < GameConfig.BALL_STOP_SPEED))
{
            if (canMove(F, N)) {
                double frictionSize = N * GamePhysicsConfig.Uk;
```

```
                double velocitySize = velocity.getSize();
                double frictionX = 0;
                double frictionY = 0;
                if (velocitySize > 0) {
                    frictionX = frictionSize * velocity.x /
velocitySize;
                    frictionY = frictionSize * velocity.y /
velocitySize;
                }
                if (velocity.x != 0)
                    F.x += velocity.x > 0 ? -Math.abs(frictionX):
Math.abs(frictionX);
                if (velocity.y != 0)
                    F.y += velocity.y > 0 ? -Math.abs(frictionY):
Math.abs(frictionY);
            }
            else {
                F = new _3dVector(0, 0, 0);
```

```
            }

        }

        return F;

    }
```

ابتدا بررسی می‌کنیم تا توپ در یکی از کناره‌ها باشد و سرعت در یک جهت ۰ باشد (اگر شرط دوم گذاشته‌نشود، در هر برخورد نیز اصطکاک اعمال می‌شود) حال در صورتی که جسم بتواند بر نیروی Us غلبه‌کند و حرکت کند، مقداری برابر با N*Uk به نسبت سرعت‌ها، از نیروی جسم کم می‌شود. اگر هم جسم امکان حرکت را نداشت، نیروی ۰ را در نظر می‌گیریم. با توجه به اینکه سرعت دقیقا 0 نمی شود، آستانه ای برای سرعت 0 تعیین می کنیم که در کد ما در config این آستانه 5 تعیین شده است.

```
    private boolean canMove(_3dVector f, double N) {

        double frictionSize = N * GamePhysicsConfig.Us;

        return f.getSize() > frictionSize;

    }
```

در این تابع، بررسی می‌شود که اندازه‌ی F، بزرگتر از N*Us باشد. چرا که در غیر اینصورت، جسم امکان حرکت نخواهدداشت.

```
    private void handleGyroscopeSensorEvent(_3dVector vec, double
deltaT) {

        theta = new _3dVector(vec.x * deltaT + theta.x,

            vec.y * deltaT + theta.y,
```

```
            vec.z * deltaT + theta.z);

    }

    private void handleGravitySensorEvent(_3dVector vec, double
deltaT) {

        theta = new _3dVector(Math.asin(vec.y /
GamePhysicsConfig.earthGravity),

            Math.asin(-vec.x / GamePhysicsConfig.earthGravity),

            Math.asin(vec.z / GamePhysicsConfig.earthGravity));

    }
```

همچنین برای محاسبه‌ی زاویه‌ی theta توسط دو نوع سنسور، از این دو تابع استفاده شده‌است. در سوال ششم، توضیحات کامل این کدها آورده شده است.

GameConfig

```
public class GameConfig {

    public enum sensor {GYROSCOPE, GRAVITY};

    public static final double BALL_WEIGHT = 0.01;

    public static final float BALL_RADIUS = 40;

    public static final int REFRESH_RATE = 16;
```

```
    public static final int ACCELERATION_FACTOR = 20;

    public static final int RANDOM_VELOCITY_HIGH = 320;

    public static final int RANDOM_VELOCITY_LOW = 160;

    public static final int BALL_STOP_SPEED = 5;

}
```

در این فایل مقادیر ثابت مورد نیاز در بازی آمده که شامل اندازه توپ و وزن آن و ... است.

GamePhysicsConfig

```
public class GamePhysicsConfig {

    public static final double earthGravity = 9.81;

    public static final double Us = 0.15;

    public static final double Uk = 0.07;

     public static final double kineticEnergyReductionFactor = 3 /
Math.sqrt(10);

}
```

در این فایل مقادیر ثابت مورد نیاز در فیزیک بازی آمده که شامل g، ضریب کاهش انرژی جنبشی (که برای هر 4 دیواره تاثیرگذار است) و مقادیر μ است.

_3dVector

از این کلاس برای پیاده سازی بردارهای سه جهتی استفاده شده است که البته در این پروژه ما تنها از دو جهت x,y استفاده می کنیم (زیرا چرخشی که خواسته مسئله است چرخش azimuth است).

پروفایل کردن اپلیکیشن

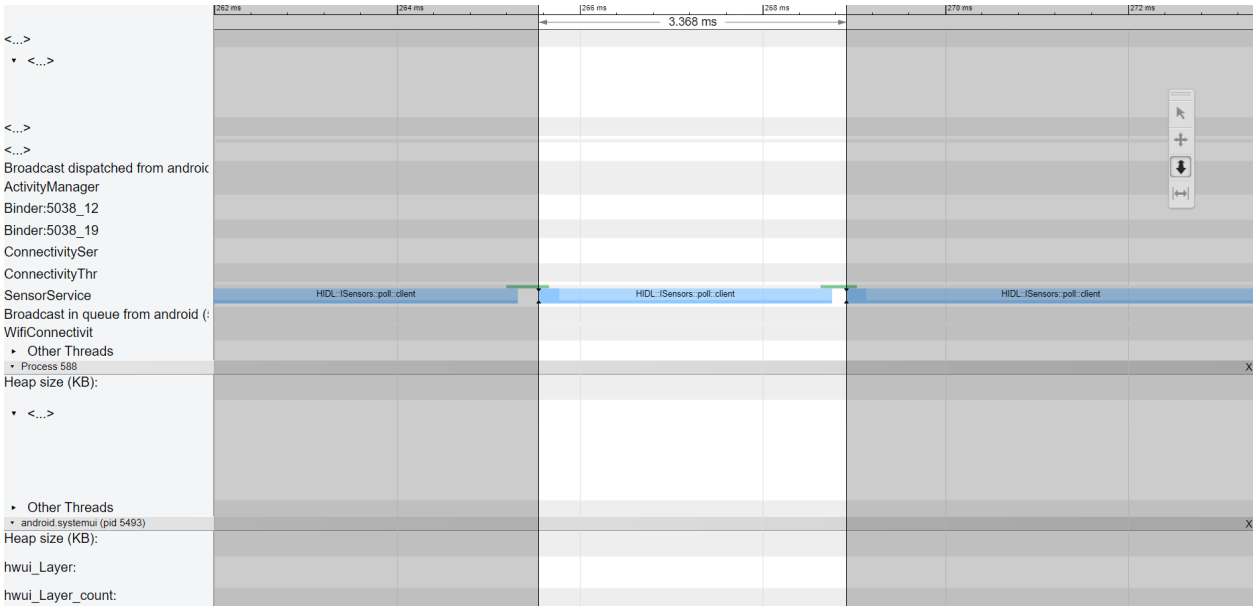
برای پروفایل کردن اپلیکیشن از ابزار systrace روی گوشی Samsung Galaxy A21s استفاده شده است. اسکریپت اجرایی نیز به صورت زیر است:

```
python systrace.py -o trace.html sched freq idle am wm gfx view \
binder_driver hal dalvik camera input res
```

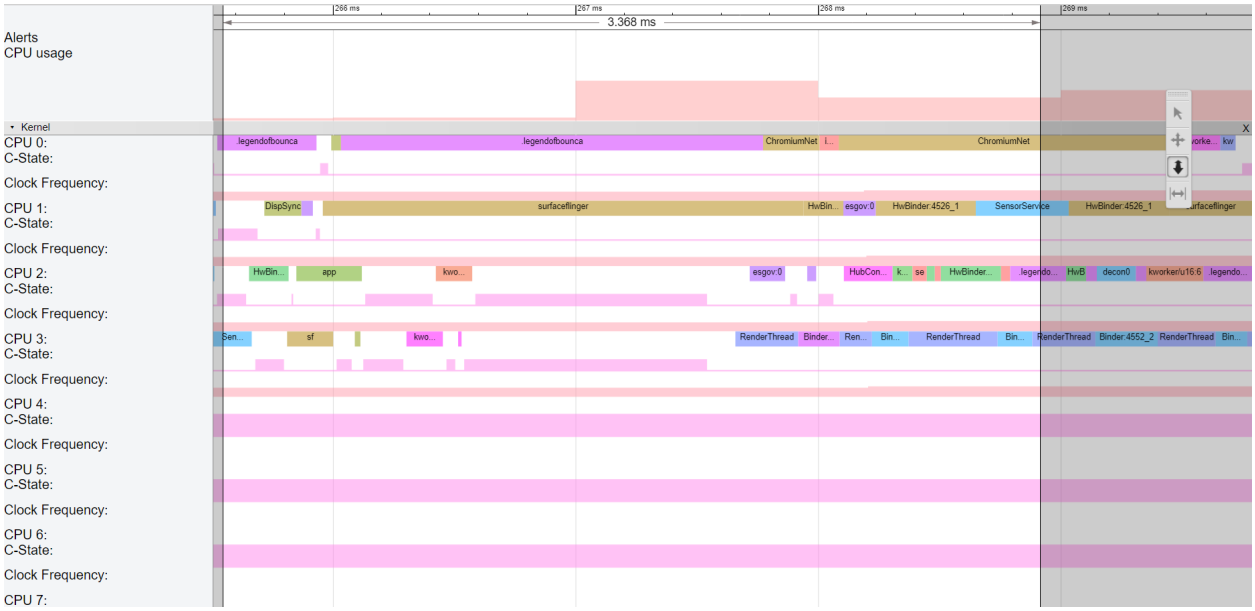
خروجی این دستور فایل html حجیمی می شد که با مرورگر باز نمی شد. برای باز کردن آن از ابزار tracing مرورگر chrome (به آدرس chrome://tracing) کمک گرفته ایم.

پاسخ سوالات

پاسخ سوال 1

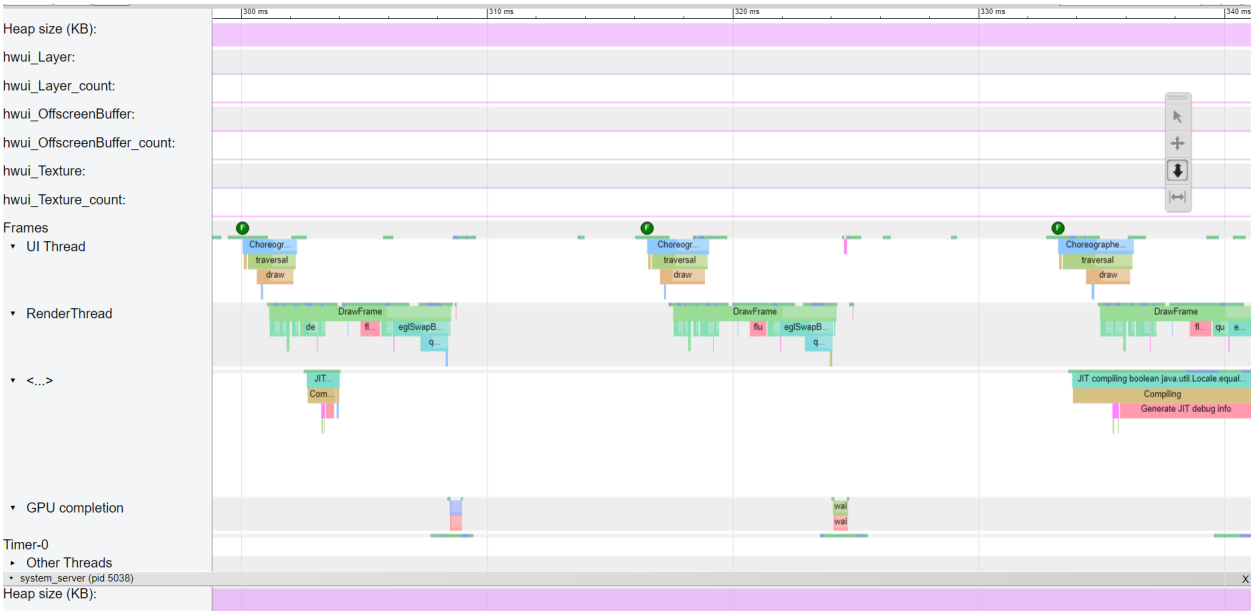


همانطور که در تصویر مشاهده می شود، اطلاعات از سنسور هر 3 میلی ثانیه (بین 3 تا 6 متغیر است) گرفته می شود. حال در این میان اتفاقات زیر رخ می دهد:

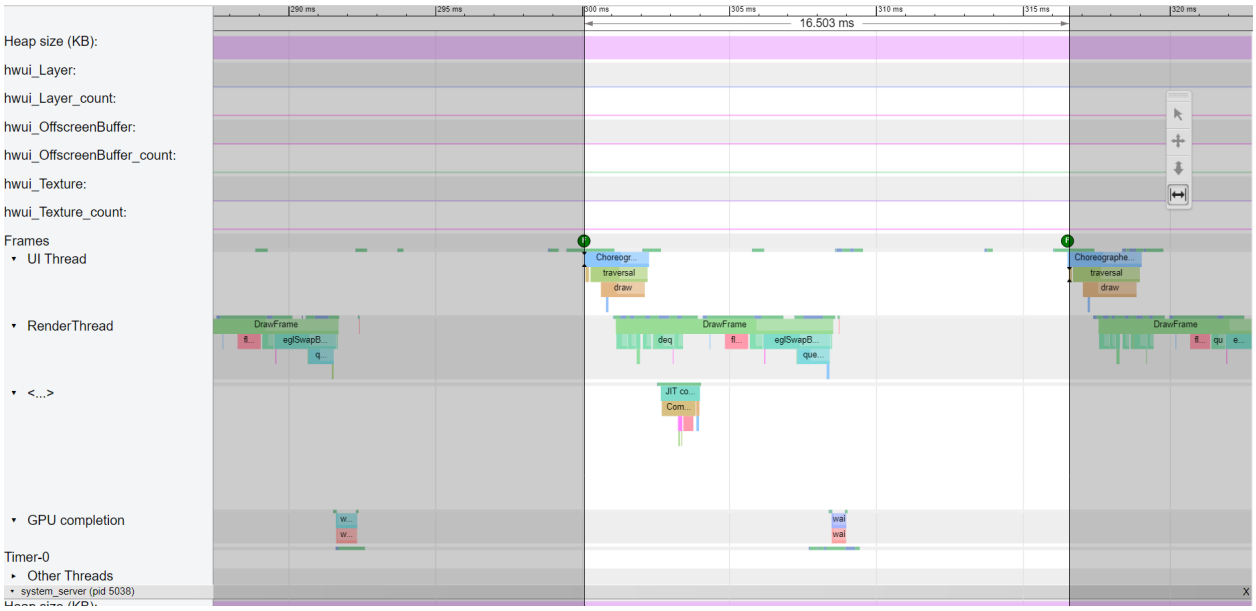


بسیاری از این اتفاقات، مربوط به نمایش اجزای صفحه و ... هستند. مثلا surfaceflinger پردازه ای است که تمامی اجزاء را برای نمایش روی صفحه نمایش buffer می کند. یک نمونه از موارد مرتبط به سنسور HWBinder است که با رنگ سبز نمایش داده شده است. این پردازه اطلاعات سنسورها را از لایه فیزیکی به لایه اپلیکیشن انتقال می دهد.

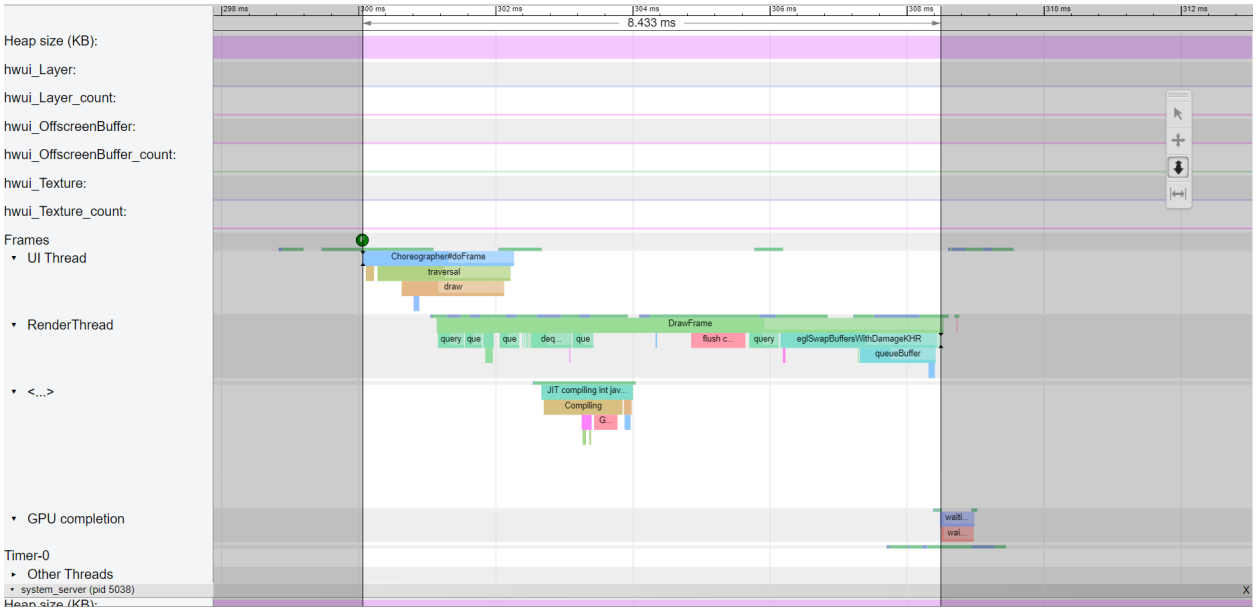
پاسخ سوال 2



همانطور که مشاهده می شود عمل رسم توپ روی صفحه نمایش به صورت متناوب انجام می شود. حال در ادامه زمان میان دو رسم توپ را مشاهده می کنیم:



همانطور که مشخص است، این زمان حدود 16 میلی ثانیه است. این زمان همان زمانی است که صفحه نمایش با آن به روز رسانی می شود (60Hz) و ما نیز زمان فراخوانی تابع به روز رسانی مکان توپ را همین 16 میلی ثانیه تعیین کرده ایم. در ادامه می بینیم که رسم توپ روی صفحه چقدر طول می کشد:



زمان نمایش توپ روی صفحه متغیر و حدود 8 میلی ثانیه است که این عمل (رسم توپ روی صفحه نمایش) هر 16 میلی ثانیه یک بار رخ می دهد.

پاسخ سوال 3

با توجه به اینکه نرخ به روز رسانی صفحه نمایش اکثر گوشی ها 60 هرتز است، اگر این تایمر را کمتر از 16 میلی ثانیه قرار دهیم بی فایده خواهد بود. اما علاوه بر این، باید توجه داشته باشیم اگر به روز رسانی مکان توپ بیش از حد سریع باشد، سربار زیادی روی پردازنده گوشی خواهد داشت و اگر به روز رسانی کند باشد، پرش تصویر مشاهده می شود. ما در این پروژه زمان به روز رسانی را همین 16 میلی ثانیه قرار داده ایم.

پاسخ سوال 4

Android NDK[1]

NDK مجموعه ابزاری است که به توسعه دهندگان این امکان را می‌دهد تا از کدهای نوشته شده به زبان‌های برنامه‌نویسی C و C++ به صورت مجدد استفاده کرده و از طریق JNI[2] آن را در برنامه خود قرار دهند.

مزایای آن به شرح زیر هستند:

- برای عملیات فشرده پردازنده و اجرای برنامه‌های محاسباتی فشرده مانند بازی‌های ویدیویی موبایل، پردازش سیگنال و شبیه‌سازی فیزیک مناسب است.
- با استفاده از آن می‌توان کد موجود C / C ++ را به Android منتقل کرد.
- نظر به اینکه برنامه به طور مستقیم در پردازنده اجرا می‌شود (به جای تفسیر توسط Dalvik Virtual Machine) سرعت فوق العاده بالایی پیدا می‌کند.
- کدی که با C / C ++ برای Android نوشته شده است می‌تواند به راحتی در سیستم عامل دیگری مانند iOS یا Windows منتقل شده و اجرا شود. بنابراین برای ایجاد برنامه‌های چند پلتفرمی مفید است.

اما نکته‌ای که در مورد NDK وجود دارد این است که پیچیدگی برنامه را افزایش می‌دهد، که این می‌تواند به محدود شدن عملکرد برنامه منجر شود.

[1] Native Development Kit
[2] Java Native Interface

Android SDK[3]

SDK مجموعه ابزاری است که از زبان برنامه نویسی جاوا استفاده می‌کند و شامل نمونه پروژه ها، ابزارهای توسعه و محیط توسعه یکپارچه[4] اندروید استودیو است. همچنین تمام API های رایج مورد استفاده برای برنامه‌های Android را فراهم می‌کند.

- مستقل از معماری پردازنده، از قابلیت حمل دستگاه اطمینان حاصل می‌کند.
- مجموعه‌ای غنی از کتابخانه‌ها را فراهم می‌کند.
- مدیریت خودکار حافظه را انجام می‌دهد.

همچنین شایان ذکر است که برخی از برنامه‌های Android از NDK برای دستیابی به عملکردی خاص استفاده می‌کنند. در نتیجه این باعث می‌شود NDK و SDK در برخی موارد به نوعی مکمل یکدیگر باشند. با این وجود، اندروید همچنان توصیه می‌کند که فقط در صورت نیاز به استفاده از NDK بپردازیم.

پاسخ سوال 5

سنسورهای مبتنی بر سخت‌افزار[5]: این سنسورها اجزای فیزیکی هستند که در یک گوشی یا دستگاه تبلت تعبیه شده‌اند. سنسورهای سخت‌افزاری داده‌های خود را با اندازه گیری مستقیم خصوصیات خاص محیطی مانند شتاب، قدرت میدان ژئومغناطیسی یا تغییر زاویه بدست می‌آورند.

سنسورهای مبتنی بر نرم‌افزار[6]: این سنسورها دستگاه های فیزیکی نیستند، هرچند که از سنسورهای مبتنی بر سخت‌افزار تقلید می‌کنند. سنسورهای مبتنی بر نرم‌افزار، داده‌های خود را از یک یا چند سنسور مبتنی بر سخت‌افزار می‌گیرند و گاهی اوقات سنسورهای مجازی یا سنسورهای ترکیبی نامیده می‌شوند. سنسور مجاورت و سنسور شمارنده گام نمونه‌هایی از سنسورهای مبتنی بر نرم‌افزار هستند.

حال به بررسی سنسورهای به کار رفته در این تمرین می‌پردازیم:

---

[3] Software Development Kit
[4] IDE
[5] Hardware-based sensors
[6] Software-based sensors

★ سنسور Gravity در دسته‌ی سنسورهای مبتنی بر نرم‌افزار قرار می‌گیرد.

★ سنسور Gyroscope در دسته‌ی سنسورهای مبتنی بر سخت‌افزار قرار می‌گیرد.

پاسخ سوال 6

سنسور Gravity : این سنسور نیروی جاذبه‌ی وارد بر یک وسیله را بر روی هر سه محور فیزیکی (x, y, z) بر حسب m/s^2 اندازه می‌گیرد. معمولا از آن برای تشخیص حرکت، لرزش و شیب استفاده می‌شود.

سنسور Gyroscope : این سنسور سرعت چرخش یک وسیله را حول هر سه محور فیزیکی (x, y, z) بر حسب rad/s اندازه می‌گیرد. معمولا از آن برای تشخیص چرخش استفاده می‌شود.

برای کوتاهی و یکدست بودن کد، در هر دو روش، این نیروها به زاویه تبدیل شده‌اند. در سنسور ژیروسکوپ، مقدار خوانده‌شده از سنسور، مطابق فرمول داده‌شده در deltaT ضرب‌شده و به مقدار قبلی اضافه شده‌است.

در سنسور جاذبه، مقدار خوانده‌شده توسط سنسور، تقسیم بر گرانش زمین شده و از آن arcsin گرفته شده‌است (دلیل این کار، همانطور که توضیح داده شد، تبدیل کردن شتاب به زاویه می‌باشد).

در سنسور جاذبه، محورهای x و y، با محورهای x و y در سنسور ژیروسکوپ متفاوت می‌باشد. به همین دلیل، در سنسور جاذبه، این مقادیر جابجا شده‌اند. همچنین قسمت منفی و مثبت محور x نیز جابجا می‌باشد که باعث قرینه‌شدن محور x در محاسبات شده‌است.

پاسخ سوال 7

اگر بازی روی سطح شیب دار شروع شود، در استفاده از سنسور gravity، توپ به سمت پایین (به سمت گرانش) سقوط می کند؛ در حالی که در استفاده از سنسور gyroscope توپ حرکتی نمی کند. علت آنست که در سنسور gravity، اطلاعات لحظه ی کنونی به اطلاعات لحظه ی قبل ربطی ندارد اما در سنسور gyroscope اطلاعات لحظه ی کنونی وابسته به تغییرات آن نسبت به اطلاعات لحظه ی قبلی است. بنابراین با شروع بازی روی سطح شیبدار، سنسور gyroscope حرکتی برای توپ ایجاد نمی کند در حالی که سنسور gravity بخاطر عدم وابستگی به اطلاعات قبلی، برای توپ حرکت ایجاد می کند.